ENCRYPTION AND DECRYPTION OF DIGITAL MESSAGES IN PACKET TRANSMITTING NETWORKS

Field of the invention

The present invention refers to a system for encryption and decryption of messages in packet transmitting networks.

Background

Over the past few years the Internet has expanded enormously in terms of traffic volume as well as in terms of the extension of the network. Higher transmission speeds are continuously pursued by the market in order to be able to distribute multimedia, games, video, HDTV and other media forms. A large bandwidth is required to be able to transfer these media forms in a digital format. Furthermore, the value of the video movies, the games, etc. distributed over the Internet is very high. On the whole the expansion of the Internet has entailed that very sensitive confidential information is today transmitted over the Internet and other packet transmitting networks or packet switching networks without a satisfactory protection. This fact has created a need to protect the information that is transmitted over these networks from unauthorised access, such as eavesdropping, copying and manipulation. Encryption can be used to protect information transmitted over the networks against such unauthorised access.

The underlying technology of the Internet is based on packet transmission. The basis of packet transmitting networks is found in technologies such as IP (Internet Protocol) and other packet transmitting technologies such as ATM (Asynchronous Transfer Mode) [ITU I.321]. Encryption can be applied at different levels in the protocols constituting the Internet or other IP based networks. The lowest level at which encryption can be applied is the IP-level according to [RFC 791, Jon Postel "Internet Protocol", RFC 791, September 1981]. Other higher encryptable levels are for example the TCP (Transmission Control Protocol) level according to [RFC 793, Jon Postel, "Transmission Control Protocol", RFC 793, September 1981]; the UDP (User Datagram Protocol) according to [RFC 793] level;

the IPSEC according to [RFC 2406, Kent, S., and Atkinson, R. "IP Encapsulating Security Payload, (EPS)", RFC 2406, November 1998]; the SMTP according to [RFC 821, Jon Postel, "Simple Mail Transfer Protocol", RFC 821, August 1982]; the Secure Sockets Layer [RFC SSL, The SSL Protocol Version 3.0, Transport Layer Security Working Group, Alan O. Freier, Philip Karlton, Paul C. Kocher, November 18, 1996]; or other functionally corresponding levels in the complete logical protocol independent of physical form of transmission.

The use of packet transmitting networks of a different nature such as WAP (Wireless Application Protocol) for mobile applications also require that the security aspects are solved in an adequate way by encryption, since unauthorised eavesdropping of transmission otherwise becomes possible. The security can in this connection be located either at a lower or at a higher level of the protocol. So, for example, the transaction layer in WAP according to WTP (Wireless Transaction Protocol) contains several different types of transaction focussed services such as the security layer WTLS (Wireless Transport Layer Security), which are security functions based on the security sockets layer SSL[RFC-SSL]. The transport layer WDP (Wireless Datagram Protocol) is a transport layer protocol which is able to operate over many different types of underlying protocols. This level may in a simplified manner be described as a unification of the IP and the TCP from the Internet protocols. Further information is found in [RFC 241 R. Thayer, N. Doraswamy, R. Glenn, IP Security Document Roadmap, November 1998].

Encryption and decryption of information has for a long time been an important tool for preventing unauthorised and unwanted access to secret information, regardless of whether this information is stored in a computer, stored on a storage medium that is readable by a computer or transmitted between two parties over a communications link. With the development of computers and telecommunications technology, the amount of information that is created and exchanged on a day-to-day basis is ever increasing and becomes more easily available. The need to prevent unwanted access to and possible manipulation of this information in a manner that is easily implemented and insures a high level of security is therefore larger than ever before.

By encryption, a message or a communication link is protected against unauthorised access or corruption by converting a message to a cryptogram according to a selected conversion function. Usually a function is selected from a family of functions dependent on a so-called cipher key. In the terminology of the trade the expression cipher system refers to a description of a construction of the selected family of cipher functions. For the purpose of the present patent application a cipher system means such a description in the shape of apparatuses and methods that together specify the invention.

It is a difficult task in several aspects to select a good cipher system. Since the consequences of selecting a bad cipher system can be disastrous, for example in the case that important business information falls into the hands of unauthorised people, the cipher systems are usually subject to a number of rules with regard to its construction and use. Within the frame of these rules, there are also such aspects as interoperability between different system parts and other more technical problems. Sometimes these rules are designed in the shape of a national or an international standard. It may for example concern mobile communications networks or the networks for the electronic transactions of the banks.

The rapid development within the computer technology during the last few years has given the cryptanalyst the possibility of an extremely fast statistical processing of encrypted messages, causing cipher systems that earlier have been considered safe to be cracked. This fact has considerably increased the demands on the resistance against breaking of the cipher systems. Similarly, the development of programmable electronics has increased the possibilities of cryptanalysts to gain unauthorised access of encrypted information. The logical conclusion would be to replace such cipher systems that are out of date, in a sense that they are comparably easy to break, for safer cipher systems. However, for many applications this is in practise not feasible, since large systems and many involved operators restrain the change of cipher systems for financial and other reasons. For example, the possibilities for a single operator in a communications network to replace the cipher system is directly constrained by the fact that all operators have to use the same cipher system according to the presently valid standard. There may also be other

more formal obstacles, for example legislation or policy rules controlling what cipher systems are to be used. With regard to the financial aspect the costs of changing an existing cipher system for another and better system may be unacceptably high. In particular, costs arise due to the exchange of a large number of physical cipher units, which perhaps in addition would have to be specifically tested and approved.

Encryption and decryption schemes typically rely on the use of an algorithm in combination with a data sequence or a so-called cipher key. Conventionally, symmetric algorithms, wherein the sender and receiver (or creator and reader) of information share the same secret key, are most widespread. These schemes generally fall into one of two classes, stream ciphers and block ciphers. An example of the latter scheme is the Data Encryption Standard (DES) described in US-A-3 962 539. In such a scheme, the algorithm is time-invariant. In other words two different messages (plaintext) encrypted with the same key will undergo an identical series of computational steps. Depending on the algorithm, a change of key may alter the computation only slightly. Since no algorithm and key combination is truly safe (a cryptanalyst attempting to decrypt a message armed with a powerful computer is limited only by the time required to try all possible permutations) and the algorithm of known schemes is essentially invariant, the security of an existing system often relies on the frequent, often daily, change of keys.

Prior art

The problem with the use of different technologies for encryption in packet transmitting networks is that the traditional methods such as DES (Data Encryption Standard) [FIPS 46; US-A-3,962,539] or IDEA [PCT/CH91/00117, 28 Nov 1991; US 5,214,703, 25 May 1993] is that they are to slow for wideband applications at encryption speeds in the magnitude of for example 2-10 gigabit per second.

Another problem in wideband communication is the large volumes of data gathered by the cryptanalyst. It is a well known fact that the cipher key becomes statistically worn out when a sufficient amount of traffic can be gathered for analysis, and the cipher or the cipher key may thereby be more easily broken.

Traditional cipher systems, such as the above mentioned systems, are designed to handle a smaller amount of data than the amount required in a wideband system that should be able to handle speeds of for example 2-10 gigabit per second. As an example it can be mentioned that at a transmission speed of 6.4 gigabit per second the total amount of data transmitted during 24 hours is 552960 gigabit.

Furthermore, the time required for changing the user in a time shared system (for example a server) that handles many users simultaneously also causes difficulties since the time for changing the user is substantial in relation to the performed work.

In general, the block length in traditional encryption methods has a fixed length, which cannot be changed without substantial difficulties and which often are poorly adapted to the optimum length for the transmission packet in packet transmitting networks.

Various schemes have been proposed to alleviate some of the disadvantages of prior art arrangements by increasing the complexity of any given algorithm. An example is the scheme described in US-A-5 742 686 to Finley. This document suggests a device and method for dynamic encryption wherein different encryption and decryption programs are selected and executed optionally repetitively on the basis of a stored data set, which serves as the cipher key. While this prior art scheme allows the creation of custom encryption and decryption codes on a per user basis, the encryption algorithm used by each user will be invariant, and the complexity of this algorithm will depend directly on the strength and number of encryption and decryption codes utilised.

In US-A-5 365 589 to Gutowitz a scheme for encryption, decryption and authentication is described that utilises dynamical systems, that is, systems comprising a set of states, and a rule for mapping each state forward in time to other states. The dynamical systems employed are cellular automata. A collection of cellular automata are used as secret keys for the system. Initially a subset of this collection is selected for encryption, and the message to be encrypted is encoded into the current states. The selected keys are applied over a predetermined number of cycles and the resulting current states constitute the ciphertext. While this scheme

is based on cellular automata it may not be considered truly dynamical because the number of iterations of the rules of the current key or keys is a fixed quantity determined in advance. The complexity is dependent on the number of keys or rules applied in any current encryption, and whilst it is possible to apply several rules in any single encryption, this is not always practicable. Furthermore, the inclusion of some reversible dynamical systems as current keys may introduce a weakness in the system.

Other examples of prior art are given by the documents EP 0 406 457 A1, US 4,157,454, EP 0 759 669 A2, US 4,608,456 and Kaliski B.S. Jr et al.: On differential and linear cryptanalysis of the RC5 encryption algorithm" ADVANCES IN CRYPTOLOGY – CRYPTO '95. 15TH ANNUAL INTERNATIONAL CRYPTOLOGY CONFERENCE PROCEEDINGS OF CRYPTO '95 SANTA BARBARA, CA, USA, 27-31 AUG 1995, ISBN 3-540 60221-6, 1995, BERLIN, GERMANY, SPRINGER-VERLAG, GERMANY, PAGES 171-184 XP-002096305.

Object of the invention

It is thus a general object of the invention to provide a system and a method permitting secure encryption in packet transmitting networks.

Different aspects of the problem is to achieve an encryption corresponding to the requirements on transmission and encryption speed in such packet transmitting networks; and to achieve an encryption which is resistant against statistical processing of large amounts of encrypted information.

Summary of the invention

According to the invention the object is achieved by a system for encryption according to the following description, which system is devised to enable encryption of varying packet length. The safety against statistical processing of encrypted information is met by the fact that the encryption algorithm is undefined and varied dependent on the plaintext information that is to be encrypted. The invention further provides the possibility to implement an encryption system that

fulfils the requirements of encryption speed of the packet transmitting systems. The invention also permits the change of a user or an encryption channel without a time delay.

According to one aspect of the invention there is provided an arrangement for converting information from a first format into a second format, comprising

- a memory for storing data,
- means for updating the memory with input information,
- an instruction table comprising a set of operations adapted to modify the state of the memory,
- processing means adapted to select operations from the instruction table in response to at least part of the input information and to execute the selected operations on the contents of the memory,
- at least one of the set of operations being selectable in response to any possible configuration of at least part of the input information, and
- means for extracting output information from the memory.

According to another aspect of the invention there is provided a method for the conversion of information from a first format into a second format comprising:

- establishing a set of operations for modifying the state of a memory,
- storing input information in a first format in the memory,
- selecting operations from the set in response to at least part of the input information and executing the operations on information stored in the memory, wherein the set of operations is devised such that an operation will be selected in response to any possible input information stream, and
- extracting information from the memory in a second format after executing at least one operation.

A characteristic of the method and apparatus according to the present invention is that the process by which the input information is encoded depends entirely on this input information. Specifically, both the sequence and the number of operations executed is defined by the information to be encoded. The input information essentially serves as a program for its own encryption. Consequently, the process cannot be described in terms of an algorithm because by definition it

must differ for each different input information stream. Even with knowledge of the structure of the arrangement or the steps of the method according to the invention, the actual process executed will be indeterminate until information is actually supplied. This has the advantage that the process executed cannot be described or determined without knowledge of the input information. Furthermore since each freshly selected operation will be carried out on the accumulated results of previously input information stored in the memory, even partial knowledge of the input will not facilitate reconstruction of the actual process executed because the output will be a function of all the input.

A further advantage is that inputting random information will necessarily generate a random output, since both the operations executed and the information on which the operations are performed will be random.

In a further aspect of the invention the aforementioned arrangement is included in a system for the encryption and decryption of message data comprising a cipher device, the cipher device being adapted to receive message data and at least one cipher key and to generate encrypted data corresponding to an encryption of the message data, wherein the output data extracted from the data processing arrangement is the cipher key.

In a still further aspect of the invention the aforementioned method is utilised in a method for the encryption and decryption of message data including utilising the information extracted from the memory as cipher key and encryption of the message data with a cipher function and the cipher key to generate encrypted information.

By utilising the arrangement and the method according to the invention as a cipher key generator for a cipher primitive, the overall strength of the cipher primitive can be substantially increased. Not only will the generation of a key be a highly complex process but, in addition, several different keys can be generated and used for the encryption and decryption of a single message. This permits the security of any known cipher system to be substantially improved and has the added benefit of eliminating the need for the communicating parties to have access to a large collection of shared keys.

Definitions

In the present description the established terminology in the technical field is used, such as plaintext meaning an information message that is not encrypted, and ciphertext for an information message that is encrypted according to any encryption function. Plaintext and ciphertext, respectively, refer to an arbitrary form of information or data, for example text, numerical information, image information, number signals, control or communication signals.

In the description, the expression communicatively coupled is used to describe, independently of the implementation, the coupling between two units for the exchange of data signals through a signal connection or a databus as in a hardware implementation, or parameter values or other information between parts of a software program in a software implementation.

Brief description of the drawings

Further objects and advantages of the present invention will be apparent from the following description of the preferred embodiments that is given by way of example with reference to the accompanying drawings, in which:

Fig 1-7 show embodiments of the technology upon which the invention is based;

Fig 8 and 9 show block diagrams schematically depicting encryption and decryption, respectively, of information packets in the form of data blocks according to the so-called bump-in-the-wire method,

Fig 10 and 11 show block diagrams schematically illustrating encryption and decryption, respectively, of information packets in the form of data blocks according to the so-called tunnelling method.

Detailed description of embodiments of the invention

Fig 8 shows schematically a first cipher unit 210 arranged in accordance with the description accompanying fig 1-7. The cipher unit 210 is provided with a plaintext input for inputting an information packet to be encrypted, a cipher key

input 212 for inputting a cipher key and a ciphertext output 213 for outputting an encrypted information packet. An information packet or an information cell 220 arriving at the cipher unit 210 typically comprises a block head 221 and a plaintext data block 222. The plaintext data block often has a variable length.

The arrangement according to the invention generates after encryption an output data block 230 comprising a block head 231 and an encrypted data block 232. The embodiment shown in fig 8 discloses the invention arranged to encrypt according the so-called bump-in-the-wire method, which means that only the data block 222 is encrypted while the block head 221 is transferred unencrypted to the outgoing block head 231.

In order to be able to decide which cipher key that is to be used, a look-up function 240 reads the block head 221 of the arriving block and then performs a look-up in the security database 250. A selected data record 251 is read from the security database 250 to the look-up function 240, which then extracts a cipher key from the data record 251, the cipher key then being transferred to the cipher unit 210. The cipher unit 210 then carries out encryption of one or more data blocks dependent on the selected cipher key.

The look-up function 240 further attends to that the possible necessary changes of the block head 221 are carried out. Such changes may consist of a change in the address information in the block head, which possibly is controlled by address information together with the cipher key in the data record 251. After the encryption of the plaintext block 222 to the corresponding ciphertext block 232, the block head 231 is concatenated with the ciphertext 232.

Fig 9 shows an inventive embodiment for the decryption of blocks that have been encrypted according to the so-called bump-in-the-wire method, as has been explained in connection with fig 8. A cipher system 210 implemented according to the description in connection with fig 1-7 is arranged to decrypt an input information packet or an information cell 230 comprising a block head 231 and a ciphertext data block 232 to an output data block 220 comprising a block head 221 and a plaintext block 222. The cipher key that is to be used in the decryption is determined by a look-up function 240 dependent on the block head 231 on the input

block by means of a look-up security database 250. A selected data record 251 is read from the security database 250 to the look-up function 240, which then extracts a cipher key from the data record 251. The cipher key is in its turn transferred to the cipher unit 210. The look-up function further attends to that possible necessary changes in the block head 231 are carried out, which in similarity with the case in encryption may comprise a change of the address information in the block head possibly controlled by the address information together with a cipher key in the data record 251. After decryption of the cryptogram 232 to the corresponding plaintext block 222, the block head 221 is assembled with the plaintext block 222 and the decryption is finished.

In fig 10 a cipher system according to the description in connection with fig 1-7 is implemented in a cipher unit 210, which encrypts one or several input information packets or information cells 220 each consisting of a block head 221 and a plaintext data block 222 to an output data block 230 consisting of a block head 231 and an encrypted data block 232. The embodiment shown in fig. 10 illustrates the invention arranged to encrypt according to the so-called tunnelling method. The tunnelling method means that one or several data blocks, including each respective block head, is encrypted and transmitted to the output block 230, which is provided with a changed block head 231. In order to be able to decide what cipher key to use, including decisions of which input blocks 220 that are destined to the same receiver and thus may be encrypted together to one single continuous encrypted data block 230, a look-up function 240 reads the input block head or block heads 221 of the block or the blocks and then carries out a look-up in a security database 250. The selected data record 251 is read from the security database 250 to the look-up function 240, the data record 251, including the cipher key that is to be used and a block head, which data record is then transferred to the block head 231 in the output block 230. The look-up unit 240 then transfers the current cipher key from the data record 251 to the cipher unit 210. An assembly function 260 is arranged just before the cipher unit 210 at its plaintext input, possibly including means for data compression. The assembly function 260 assembles the input data blocks 220, and possibly also compresses them, in such a

way that it is possible to split the blocks again after the encryption. In practical use of the invention in the embodiment shown in fig 10 there is usually a stream of input packets, all with the same sender and receiver. This is detected by inspecting the block head 221 of the input packets. It may be chosen to encrypt and send a plurality of input blocks together, since this gives a more economical utilisation of the band width.

In fig 11 a cipher system according to the description in connection to fig 1-7 is implemented in a cipher unit 210 which is arranged to decrypt an input information packet or information cell 230 comprising a block head 231 and a ciphertext data block 232 to a data stream. The data stream is then converted by an unpacking unit 262 arranged at the plaintext output of the cipher unit to one or several output data blocks 220 each comprising a block head 221 and a data block 222. The embodiment of fig 11 shows the invention arranged to decrypt messages according to the so-called tunnelling method, c.f. fig 10. In order to be able to decide what cipher key that is to be used in the decryption, a look-up function 240 reads the block head 231 of the input block and then carries out a look-up in the security database 250. A selected data record 251 is read from the security database 250 to the look-up function 240, the data record containing the cipher key that is to be used in the decryption. The look-up unit 240 then transfers the current cipher key from the data record 251 to the cipher unit 210. Immediately after the cipher unit, which here is arranged for decryption, there is an unpacking function 262 that is arranged to separate the data blocks 220 contained in the decrypted block. If a plurality of blocks are encrypted together there is a problem in a delayed transmission, which can cause trouble due to the prolonged time the receiver has to wait for the packet. At the same time there is an advantage in assembling a plurality of packets, particularly when data compression is arranged at the plaintext side preferably in the assembly function 260 and the unpacking unit 262, respectively, since data compression has an increasing relative efficiency when the blocks are longer.

An in practice implementable balance between response time and block size, which automatically adjusts the output block size as a function of the

frequency of input blocks is achieved by:

A. resetting a clock when the first packet arrives;

B. adding all further blocks that are input in the transmission and outputting in the same ciphertext block for a predetermined period of time; and by

C. breaking when the clock attains a preset value.

An advantage with this method is that long data transmissions are favoured while avoiding that the response time can be longer than a certain preset value.

Different embodiments of the invention are adapted to encryption and decryption, respectively, of information blocks with inter-alia the following content:

multimedia information;

software module or program part to an application program;

software module or program part to a game program;

part of an image stream to HDTV;

packets of the IP (Internet Protocol) type to the Internet;

packets adapted to the ATM (Asynchronous Transfer Mode)[ITU I321];

blocks according to the TCP (Transmission Control Protocol) level according to [RFC 793];

blocks according to the UDP (User Datagram Protocol) according to [RFC 793];

blocks according to the IPSEC [RFC 2406];

blocks according to the SMTP [RFC 821];

blocks according to the Secured Sockets Layer [RFC SSL];

blocks according to the WAP (Wireless Application Protocol);

blocks in the Transaction layer, WTP (Wireless Transaction Protocol);

blocks in the Security layer, WTLS (Wireless Transport Layer Security); or

blocks according to the WDP (Wireless Datagram Protocol).

<u>Detailed description of the encryption functionality</u>

A machine or an apparatus module embodying the present invention is shown in Fig. 1 by reference numeral 10. The module 10 comprises a processor 11, a program memory 12, an instruction memory or look-up table 13, and a general

purpose memory 14, which preferably should be accessible at least partially randomly. An input port 16 and output port 17 are provided for inserting and extracting information, respectively, and an output register 15 is included between the general purpose memory 14 and the output port 17.

The input information, which in the preferred embodiment is in the form of a binary string, is input into the program memory 12. This memory 12 preferably has a large capacity to enable as much as possible of the input information to be accessed as program, as will be explained below. The instruction table 13 holds a predetermined set of instructions or operators (op-codes). These operators are addressed using sections of the input information stored in memory 12 as addresses or indexes to the table 13. Hence the input information essentially serves as a program according to which the processor 11 executes the operations of the instruction table 13 on data in the memory 14. The number of operators stored in the instruction table 13 is chosen to correspond to the size of the input information sections serving as program steps, so that every possible permutation of input numbers accesses a valid operator. In other words, any possible string of input numbers, in this case, any possible input binary string, is a valid program. In the exemplary embodiment, the processor 11 reads 10 bits of the stored input information at a time as a program step. This information could have any value between 0 and 1023. Hence to ensure that any possible input string will enable the processor 11 to select a valid operation, the instruction memory holds 1024 operators. The operators are preferably different and independent of one another so that no two different input strings will cause an identical sequence of operations.

Whilst in the embodiment described with reference to Fig. 1 the number system used for storing and manipulating the information is binary, which advantageously permits the use of a digital processor, it will be understood that any number system may be used according to the needs of the implementation of the module 10 and the application requirements.

An instruction pointer (IP) (shown in Fig. 2) is associated with the program memory 12 and used by the processor 11 to select the address of operators contained in the instruction table 13. The program steps constituted by the input

information may be selected in a step-wise fashion from one end of the information to the other, however the processor 11 is preferably capable of controlling the pointer IP to select a program step from any portion of the information input. This in turn implies that the program memory 12 must have a capacity large enough to allow access to any section of a large amount of input information. The instruction table 13 preferably permits read and write operations to enable the order of the instructions, that is the addresses of each instruction, to be changed. The instructions themselves are chosen to change the state of the memory 14 in some way. The instructions typically include, but are not limited to, fast operations such as add, subtract, table lookup and slow operations such as integer multiply and iterations. However it is important that the instructions are limited to operate on areas of memory 14 containing data and that instructions that may arrest processing are excluded. The operations in the instruction table 13 are selected to update the memory 14 and thereby change the state of the module 10 only.

In the preferred embodiment, data is processed in units of 32 bits. Accordingly, the general purpose memory 14 holds 32-bit words, and is of a size sufficiently large to ensure adequate complexity in the generation of the output data. It should be noted that to achieve maximum complexity for any fixed number of operations, full computability should be provided. This requires that the memory be extensible, that is, it should have a size that can be altered during processing to avoid the necessity of rounding-off. It should be noted that rounding-off may cause to two different operations to produce the same result and accordingly limits the possible processing diversity of the module 10.

In Fig. 1 a bi-directional connection is schematically shown between the program memory 12 and the general purpose memory 14 to indicate that data may be exchanged between the two. Specifically, when inputting data, this information can be fed both to the program memory 12 and to the general purpose memory 14. Hence input data serves simultaneously as program and operand, and the output data will contain traces of input data that has been transformed by the program. Similarly, data from the general purpose memory 14 could be pulled into the program memory 12 and used as program. It will be understood that the initial

content of the general purpose memory 14 depends on the particular application of the module 10. This will be discussed in more detail below.

The output register 15 serves to buffer blocks of output data extracted from the general purpose memory 14. The output register 15 is structured with a number of rows. In the preferred embodiment, the register 15 contains 13 such 32-bit rows. The output data is read from predetermined locations in memory. For example, if the memory were implemented as a number of stacks (see discussion below) the output data could be taken from the top of the stacks.

The extraction of output data could occur periodically, for example after the execution of a predetermined number of operators. This could be implemented using a counter and stop flag that can be updated by the processor 11. The stop flag is preferably a specific location in the general purpose memory 14. However, the extraction of output data should preferably be dependent on the input information in the same way as the processing of this information. In other words the point in time at which the stop flag will be set should be indeterminate prior to supplying the input information. Specifically, the point in time at which extraction is enabled is determined by the occurrence of a number of selected operators or the contents of a particular location in memory, or a combination of the two. This is implemented in the module 10 by providing a stop flag that can be consulted by the processor 11 at intervals, for example after every operation. The stop flag is a reserved portion of memory 14. At least one of the operators contained in the instruction table 13 updates the stop flag when it is executed. Such an update will not normally consist of setting the stop flag to "stop" but rather to assign "stop" only if some condition is met by some data. Each individual operation that updates the stop flag, are preferably devised to do this in different ways.

Preferably several of the operations will be adapted to update the stop flag. When the stop flag is set, data is output onto the output register 15 from at least one specific location in memory. The specific location or locations may be pre-defined or be dependent on the operators called. In addition, specific operations may be performed on the output data prior to its transfer to the output register 15. These additional output operations are preferably selected from a plurality of possible

operations selected according to the value of a predetermined location in the general purpose memory 14.

It will be understood that the updating of the stop flag need not be limited to the description given above. However, it is important that certain conditions are imposed on the generation of the stop flag to reduce the risk of data being extracted after undergoing only very few operations.

It should further be noted that the calling of the stop flag need not arrest processing. While in a software implementation of the module 10, it is convenient to permit the processor to consult the stop flag after the execution of each operation, it will be understood that this could be done in parallel with the execution of operations. It is further evident that if the output operation were implemented entirely in hardware, the triggering of output data extraction could be independent of the functions of the processor.

The register 15 may output data in a block of equivalent size to that of the input blocks, of a larger size, or of a smaller size.

The module 10 may also include a feedback connection between the output register 15 and the general purpose memory 12 so that output is also used as program to process the contents of memory 14. This provides added security and reduces the risk that some of the data in the output register does not change from one extraction to another. Also the final output extracted by the output register 15 will then be a function of both the information input and information output. Depending on the application of the module 10, it may be advantageous to iterate this feedback operation for at least a predetermined number of times.

Optionally there can be provided a direct input connection (not shown) to the general purpose memory 14, to enable the initial state of this memory to be set externally. It will, however, be understood that the general purpose memory 14 could be at least partially filled with initialising input data via the input port 16 and the program memory 12 under control of the processor 11.

While in Fig. 1 there are schematically depicted various individual elements and connections between individual elements of the module 10, it will be understood that the implementation of the individual functional elements and the

exchange of data between these elements may be achieved in different ways. In particular, a single random access memory could be provided for storing the input data program, the operand data and possibly also the operators of the instruction set and their addresses. In order to obtain extensible memory, it is preferable to implement the memory 14 as at least one stack and possibly also at least one register, to enable data to be transferred from stack to register or vice versa. However, full computability, i.e. the capability to simulate any possible machine, will be enabled only when at least two stacks are provided. An equivalent level of computability would be provided with a bi-directional readable and writeable tape.

In another wording, the definition of full computability in a data processing device may also be expressed as the device being capable of, given a suitable program, simulating any computational process. Such a data processing device is most often referred to as a computational machine being capable of universal computation.

Fig. 2 shows a preferred implementation of the memory elements of module 10 wherein the various interconnections are omitted. In this figure, elements similar to those shown in Fig. 1 have like reference numerals. In common with Fig. 1 the arrangement according to Fig. 2 comprises a program memory 12. A program register 121, which in the preferred embodiment has a capacity for addressing a 10-bit word, is associated with the program memory 12 and serves to hold the current program step selected by the instruction pointer IP. The instruction pointer is controlled by the processor 11(Fig. 1). As mentioned above, the size of the program memory depends on the manner in which input data is used to select operators. Ideally the program memory 12 should be large enough to store an entire input message, so that the processor may select an instruction from any part of the message. In practice this may be problematic and costly, however, to enable a reasonable simulation of full computability it is preferred that the program memory has a capacity for at least 100,000 bytes. The instruction table 13 is the same as in Fig. 1 and is adapted to hold 1024 instructions that are accessed by means of the information in the program register 121. The general purpose memory 14 of Fig. 1 is replaced by three stacks 142, each adapted to hold 32-bit words, and at least one

register 141. The implementation of the memory with stacks permits the memory to grow as the processing proceeds and accordingly substantially enables full computability. The stacks 142 could be empty initially and filled progressively with input data as the input sequence is fed into the module. Alternatively, and depending on the application, the stacks 142 could be initialised with a random number sequence or any predetermined number sequence. The register 141 serves to extend the instruction set and specifically is used to temporarily store data when the stacks are updated. With the memory implemented as shown in Fig. 2, the instructions contained in the instruction table 13 can include operations to transfer data between two stacks 142, between a stack 142 and the register 141, operations on data contained in the stacks 142 and the register 141, and an operation that may alter the order of other operations. If more than one register 141 is provided, valid operations could also include transfers between registers.

In order to ensure a high complexity in the process, it is preferred that the number of available instructions is at least of the order of 500 and that the memory 14 is at least 100,000 bytes in length. If the memory is implemented in the form of stacks and registers, it is preferred that registers with capacity of at least 150 bits and at least three memory stacks are available. One way of characterising embodiments of the invention is by the requirement that the operations in the instruction table comprise such a large number of different operations that all combinations of said instructions can be simulated only by a data processing device having full computability. The least theoretical number of operations achieving this requirement is limited to a few suitably selected different operations.

The arrangement shown in Figs. 1 and 2 represent the functional structure of the module 10 according to the present invention. It will be appreciated that this function may be implemented in a number of different ways. A hardware implementation could involve the use of a microprocessor with an associated non-volatile memory containing mapping between the selected program steps represented by the input data and the predetermined instructions, and random access memory (RAM) for storing the input program, the data used as operands and the output data. The entire module 10 could also be implemented in software. This has

the advantage that the size of the program steps and the number of available operations can be changed more easily and accordingly be adapted to any specific application. A software implementation of the module would preferably be stored on a non-volatile memory, such as the hard disc of a computer, or even on a machine-readable storage medium such as a set of diskettes, CD ROM or tape for use with any data processing machine. The program could also be made available through transmission over a telephone line, on the Internet or via other communication means, by modulating a carrier signal with information representing the program.

While it may be possible to implement the functions of the data processing module 10 on any general purpose computer, this would entail the incorporation of a number of essential modifications. In particular, in the module and method according to the present invention all possible input data strings must be interpreted as valid program and be capable of addressing a valid instruction. This is necessary to prevent unauthorised instructions from terminating execution of the program and limiting the complexity of the module's function. Most general purpose computers also have a minimal set of operations where each operation perform an atomic operation only. The present invention could use an operation list of more complex kind, where each selected operation perform a series of state changes, as compared to a single state change for the well-known general purpose computer. Furthermore, the processor must be prevented from accessing extended virtual memory. Since any selected instruction must be valid, instructions such as JUMP and MOVE must be limited to areas of real memory if they are to be authorised. Finally, all instructions that arrest processing, such as a HALT instruction, or output data on a screen or printer must be excluded from the instructions set. In this respect it is important to note that the module 10 is not intended to output data in the conventional sense as part of its normal execution. It merely operates to update internal memory. In this respect, the output register 15 can be viewed as external to the processing of module 10 as such, since it extracts selected portions of the memory at intervals during operation without influencing the contents of the memory.

The module 10 is a data processing device wherein the state generated during execution cannot be predicted prior to execution. The process evolves differently for each possible input string. In other words, the process performed by the module cannot be described by an algorithm. This function has a number of useful applications, particularly in the field of cryptography. The operation of the module 10 for some of these applications will be discussed below.

Random number generator

As mentioned above, the sequence of operations performed by the module 10 depends on the input information. Hence inputting a random number sequence, commonly called a "seed", will be interpreted as a program of random operations and consequently generate a random output.

Random number generators have many uses. An example is the generation of weekly lottery numbers. Another application would be in the area of Monte-Carlo simulations, or as a random number generator for Genetic Algorithms or Simulated Annealing. An important further application is the generating of cipher keys for the encryption and decryption of information.

The initial input sequence should be obtained from a high quality random noise source, such as the SG100 hardware noise generator manufactured by Protego Information AB, Sweden.

Prior to operation, the general purpose memory 14 of the module 10 will be at least partially filled with a random sequence. This sequence may be the seed itself which would then be loaded simultaneously into the program memory 12 and general purpose memory 14. Alternatively a separate random number may be used. This has the advantage that a larger initial sequence could be used to define the state of the general purpose memory 14 than is needed or desired as a seed. The seed would then be input into the program memory 12. For this application, the seed could be input in its entirety, and the instruction pointer moved stepwise through the sequence, selecting the corresponding instruction as it moves. As an instruction is selected, the data contained in the general purpose memory 14 will be updated in some way defined by the operation. A number of operations contained in the

instruction table 13 will also update the stop flag that is represented by a location in the memory 14. The value of the stop flag will be checked at intervals, possibly after the execution of each instruction, and when it is found to be set, data contained in specific locations in memory will be read out to the output register 15 as random output.

The module 10 should preferably be capable of generating a large number of different keys from a single seed. This may be achieved simply by repeating the program defined by the seed. Preferably, however, the program will not be limited to the steps defined in the seed but will also use other data as program. For example the contents of the general purpose memory 14 might be used. This may be implemented by automatically loading the program memory 12 from a specific location in the general purpose memory 14 once the instruction pointer IP has stepped through seed, or when the program memory is empty. As a further possibility, any data extracted as a random output could also be fed back into the program memory 12. However, to reduce the likelihood of some of the output data being unchanged between extractions, the whole process should be repeated at least once with the last generated key serving as input data before the random number output is actually produced. In this way, the amount of random output that can be generated will be limited only by the run time of the module 10.

One Way Hash Function

It will be apparent from the nature of the module 10 that its function is not reversible. In other words, the module will not generate input information from the corresponding output data. As discussed above, the output string length can be fixed while the input string length may be variable. The module 10 can thus be used as a one way hash function. Other names for this function include compression function, contraction function, message digest, fingerprint, cryptographic checksum, message integrity check (MIT) and manipulation detection code (MDC).

For this application a feedback connection between the output register 15 and the program memory 12 may not be necessary. In its simplest form, a one way hash function could be implemented by initially loading the general purpose memory 14

with a predetermined bit stream, for example alternate 1's and 0's. The message to be fingerprinted is then input into the program memory 12 and the processor executes all the operations until the message is terminated and then stop. The data contained in the output register would then be the checksum, or fingerprint, of the message.

In a further embodiment, the stop flag of the processor could be disabled and the processor be adapted to enable the output register 15 to extract information from one or more specific locations in the memory 14 only when the execution of the program is terminated. In this application, the size of the output register could be selected to provide a condensed fingerprint or checksum of the message.

If the verification of a message hash function is to be kept secret, a secret key can be used when computing the hash function. This is also known as Message Authentication Code (MAC). This assumes that the parties wishing to demonstrate and to verify the authenticity of a document share a secret key, or collection of secret keys, and some convention for selecting which key is to be used. In this case, the secret key could be used as the initial value of at least part of the general purpose memory 14, as a header to the message information, or employed to change the order of the operations in the instruction memory 13, i.e. their addresses, or a combination of any of these. Only a person in possession of the key used to generate the hash function can verify whether the message is authentic or not.

Encryption/Decryption

As already discussed above, the module 10 can be used as a key generator for a cipher system when fed with a random number sequence. In a preferred embodiment the module 10 is combined with a cipher primitive to generate a highly secure cipher function. The encryption and decryption arrangement is shown in Fig. 3.

In this embodiment, the module 10 is arranged in parallel with a further element 20 representing a cipher system. The cipher system 20 may be a simple cipher primitive such as a substitution cipher, or could embody any known block or stream cipher function. However it is preferable that the cipher system 20 utilises an

algorithm with a tried and trusted level of security. If the apparatus according to the invention is to be used as a random number generator, a stream cipher could be selected, and the encrypting sequence resulting from the stream cipher could be used as a random source.

As for any encryption and decryption scheme, a secret key shared between the person encrypting the information and the person authorised to decrypt the information must be utilised.

In the preferred embodiment described below, the cipher system 20 is a block cipher function adapted to use two keys to generate ciphertext. One key is a secret key EKEY shared between the parties; the other key IKEY is generated by the module 10. In the arrangement depicted in Fig. 3, the input message or plaintext is fed simultaneously into the cipher system 20 and the module 10. The module 10 generates an internal key, IKEY, and supplies this to the cipher system 20. The secret external key EKEY that is shared by the two communicating parties, or by parties authorised to access the encrypted information, is also input into the encryption and decryption arrangement and is used by the module 10 to generate the internal key IKEY. The cipher system 20 uses this internal key IKEY and also the external key EKEY to generate ciphertext. The ciphertext is output by the cipher system 20 on the right-hand side of the figure. The output ciphertext is also fed back from the output of the cipher system 20 to the module 10, and is utilised as program data to generate subsequent keys. In this way, the internal keys IKEY will be generated as functions of both the plaintext and the ciphertext.

A further feedback connection is provided between the output and the input into both the module 10 and the cipher system 20 to allow a message to be encrypted several times prior to storage or transmission.

Before being input into the module 10 and the cipher system 20, the message plaintext is preferably compressed using a compression function 21. Any known reliable compression function can be utilised. The compression serves to eliminate, or at least reduce, any periodic pattern in the message text sequence. This is advantageous particularly when several messages at least partially share the same format, such as a header having addresses, identification and checksum or the like,

or carry a limited range of information, such as may occur for electronic money transfers for instance. As a further precaution to ensure that no two plaintext messages will be the same, a random number is also added to the message text using a combiner 22. It will be understood that while in Fig. 3 the combiner 22 is shown as a separate element, this arrangement should be understood to indicate that the addition of random noise occurs prior to the functions performed in the module 10 and the cipher system 20. In a hardware implementation of the encryption and decryption arrangement, the combining function could be performed in either the module 10 or the cipher system 20, or even in both. As discussed above, a random number should be obtained from a high quality noise source. The combiner 22 preferably interleaves random numbers with the message text as will be described below.

Information is both read into the encryption and decryption apparatus and processed in words of 32 bits. Before information can be fed to the apparatus it is formatted into blocks. These have the generalised structure shown in Fig. 4. Since the formatting into blocks is performed prior to feeding the information into the module 10 and cipher system 20, this function is preferably performed in combiner 22.

As shown in Fig. 4, the block comprises a number of plaintext portions 120 interleaved with random noise 110. Specifically, the plaintext (DATA), preferably previously compressed, is divided into portions 120 containing a maximum of 8192 bytes. If less than 8192 bytes of information are present, a smaller plaintext portion 120 is formed. The same is true if less than 8192 bytes remain after the total (compressed) plaintext is divided into portions. However, as the encryption and decryption arrangement of Fig. 3 processes information in 32-bit words, all plaintext portions 120 must be divisible by 4 bytes. This is achieved by adding a sequence of 0 to 3 bytes of random noise to any short plaintext portion 120.

A header 110 comprising 256 bytes of random noise (N) is inserted at the front of each section. The relative sizes of the data 120 and noise 110 portions have been selected to ensure that the message to be encrypted contains at least about 3% of random noise. It will be apparent to those skilled in the art that this proportion

may be changed for certain applications depending on the level of security that is desired.

Further information, denoted by x, may be provided at the end of the block. This preferably includes a checksum for the block and may also comprise further random noise.

In the present embodiment, the number n of plaintext sections 120 per block is limited to 64. Accordingly a block can contain any value between a maximum of 512 Kbytes (i.e. 524,288 bytes) and a minimum of 1 byte of plaintext.

In the present embodiment, the external key EKEY comprises a data sequence that is a multiple of 768 bytes. The number of multiples of 768 bytes determines the number of iterations performed during encryption and decryption as will be described below. During each iteration, a new sequence of 768 bytes from the external key is fed into the apparatus as a header to the input data. The information format fed into the encryption apparatus (module 10 in Fig. 1) is shown in Fig. 5.

An initialisation vector (IV) comprising 772 bytes of random noise is also fed into the apparatus and is used during set-up for initialising the state of the module 10. It should be noted that whilst the external key EKEY may be the same for several different messages, the initialisation vector IV will be different.

An embodiment of the inventive encryption function is described with reference to Fig. 6. In this procedure and all following procedures it is assumed that the memory of module 10 has the structure and functions depicted in Fig. 2.

The input block of the form shown in Fig. 4 is typically held in a file prior to processing. This block is presented to the cipher system 20 and the module 10 with the external key EKEY in step 501. The first time the process is executed, the memory 14 of the module 10 must be filled with certain initial values. This is performed in steps 502 to 504. Firstly, part of the external key EKEY is transferred to the stacks 142 (step 502). A number of transfer operations between the stacks to further complicate the procedure may be carried out in step 503. Finally the addresses of the instruction table 13 are at least partially randomised using the

external key EKEY and the initialisation vector IV which has been previously fed into the apparatus in step 501.

The generation of the first internal key IKEY begins in step 505 when the value of an operation register referred to as op-reg in Fig. 6 is computed. This register actually refers to a particular memory location in the general purpose memory 14. The computation of its value may take the form of adding some information to the previous value of the location. The newly computed value is used to access or address one of a pre-defined number of output operations (step 506). The output operations variously select the values of specific memory locations in the general purpose memory 14, perform some operation on these values and update the result in the output register 15 as the internal key IKEY. In step 507 the selected output operation is executed and the internal key IKEY generated. Encryption of the firsts 32 bits of plaintext is then performed by the block cipher 20 using the internal key IKEY, and the first 32 bits of ciphertext is generated. It should be noted that this first 32-bit unit of ciphertext is generated using only information contained in the external key EKEY and the initialisation vector IV; this allows an identical key to be generated for decryption when the block cipher function 20 is reversed. In step 509 the top of the stacks are updated using the 32 bits of ciphertext and the first 32 bits of plaintext, which in this case is the input data constituted by the compressed plaintext and random noise headers. Since the input data comprises 256 bytes of random noise before the compressed message plaintext, the first 64 blocks of 32 bits of input data will be comprised entirely of random noise.

In this process, the ciphertext generated by the cipher system 20 is used as program data. Prior to its input into the program memory 12, a header consisting of the 768-byte external key EKEY sequence is added to the ciphertext block and inserted as program into the memory 12. The program data used by the module thus has the general format shown in Fig. 5. In step 510 the instruction pointer IP is moved to the leftmost byte of the ciphertext, and in step 511 the output operation register op-reg is updated with the 16 bits from the memory location pointed to by the instruction pointer IP. The instruction pointer IP is then moved 34 bytes back towards the beginning, i.e. backwards in time, of the input block. In step 512, the

value of the operation register op-reg is then used to select one of the 1024 operations from the instruction table 13. The op-reg actually contains 32 bits, but only 10 bits are used to call an operation. The stop flag is then checked in step 513 and if it is not set, the process returns to step 511 to update the operation register and move the instruction pointer back a further 34 bytes towards the beginning of the input (ciphertext) block. The next operation of the instruction table 13 to be executed will then be selected based upon the contents of the operations register. This process continues until the stop flag is set. A mechanism is also provided to prevent the IP to access an location outside the input string. In one embodiment this is realised by setting the stop flag also on this condition thereby breaking the loop.

As mentioned above with reference to the module 10, a pre-defined number of operations in the instruction table 13 are adapted to update the stop flag. Once the stop flag is set, the pointers to the plaintext and ciphertext are moved one step, that is, into the next 32 bits of plaintext and ciphertext (step 514). This step corresponds to the next 32-bits of plaintext and ciphertext being loaded into the general purpose memory 14 and program memory 12, respectively. If the entire plaintext input block has not been encrypted (step 515), the process returns to step 505 to generate the next internal key IKEY for encrypting the next 32-bit ciphertext sequence. Otherwise, the process continues to step 516 for checking whether there is a new 768-byte key sequence $EKEY_{i+1}$. If so, the process continues at step 501 with the next $EKEY_{i+1}$, and otherwise, the process continues to step 517 to output the plaintext/ciphertext block.

As mentioned above, the block cipher 20 may comprise any conventional cipher primitive that uses substitution tables and various operations at least partially defined by the two keys EKEY and IKEY. Unlike the function of the module 10, which can only be performed in one direction, the block cipher function is reversible, provided the appropriate reverse mapping tables are used and the identical key provided. Accordingly, the decryption of ciphertext using the arrangement of Fig. 3 can also be performed using the procedure illustrated in Fig. 6.

It should be noted that the list of secret external keys $EKEY_1$, $EKEY_2$, etc., will be known to persons authorised to decrypt the information.

The initialisation vector IV is likewise known, and may even be published. Since the actual operations performed on the memory contents are dependent on the input sequence, and this input sequence by definition will contain some unknown element, a cryptanalyst will have no way of deducing the encryption function from the initialisation vector only.

In fact decryption of an encrypted message requires that the IV vector be known prior to decryption. The IV is normally sent "in clear" together with the cryptogram. It should be noted that, as the IV as well as the EKEY enters the invention both as program specification and also as input data, to the module 10, that all forthcoming operations and data, internal to the memory 14 as well as present in the output 17, will depend on these inputs.

If the IV is selected truly randomly prior to encrypting the plaintext, no two encrypted messages will share the same IV. If each iteration, according to Fig. 6, is executed with an independent external key $EKEY_i$ it is clear that the combination of an $EKEY_i$ and an IV will occur only once. The IV is the same for all iterations where the EKEYs will be different and the IV will change to next message where the EKEYs will be the same.

Both the external key EKEY and the initialisation vector IV are identical for encryption and decryption. Hence the execution of steps 501 to 507 for each iteration with a new 768-bit EKEY sequence will give the same result. For decryption it should be assumed that the block decryption performed in step 508 will be the inverse of the encryption function and will result in the generation of 32 bits of plaintext from the first 32 bits of ciphertext. The remaining steps 510 to 513 are executed as for encryption. It should be noted that the order of the external keys EKEY used will be reversed for decryption.

A schematic of the encryption and decryption for a single iteration using a cipher function and a key IKEY generated by the module 10 is given in Fig. 7. Here it is apparent that a first internal key $IKEY_0$ used to generate the first 32-bit unit of ciphertext from the first 32-bit unit of plaintext is a function of the external key

EKEY and the initialisation vector IV. The module 10 uses this first unit of ciphertext and the first unit of plaintext in addition to the external key EKEY and the initialisation vector IV to generate the second key $IKEY_1$ which is used in the cipher system 20 to generate a second unit of ciphertext from the second unit of plaintext. This process continues until all the plaintext has been processed. The final key $IKEY_{n+1}$ used to encrypt or decrypt the final units of plaintext and ciphertext will be a function of all the previous (0 to n) units of plaintext and ciphertext.

It is apparent from the schematic of Fig. 7 that the complexity of encryption increases for each unit, because the information content of the internal key IKEY becomes a function of all previously input plaintext and generated ciphertext. However, while the encryption of the first unit may be relatively weak owing to the relative simplicity of the key IKEY, this can be mitigated by iterating the encryption of the whole message using several different keys EKEY.

Furthermore, by reversing the order of the units for each iteration, the nominal strength of encryption of each unit will be equivalent, as each unit of ciphertext will be generated using keys IKEY that in total comprise information from the whole plaintext/ciphertext block. This implies that in Fig. 6, box 514, the pointers of ciphertext/plaintext blocks could move in either direction. In 511 the move of the IP pointer will always be backwards.

Once the full input information has been recovered, the correct decryption can be checked using the checksum tagged on the end of the input block. The 256 bytes of random noise header is then separated from the message information and the message information decompressed if it had been initially compressed.

Since the pointer to the plaintext/ciphertext block is incremented or decremented in the plaintext or ciphertext in units of 32 bits, the first 64 operations on plaintext will actually be performed on the random noise N header (see Fig. 4). This ensures that the initial contents of the general purpose memory 14 comprising the stacks 142 and registers 141 will be filled with random information before the first 32-bit word of plaintext/ciphertext is loaded into program memory 12.

Due to the observed structure of most input strings the present embodiment of the invention actually process the input plaintext block (Fig. 4) in the right-to-left

direction during the first iteration of the process according to Fig. 6., as this would be a security advantage for these inputs, and be of no significance to all other possible input strings.

Whilst random noise is simply placed at the head of each data sequence as shown in Fig. 4, in a further embodiment of the invention this random noise is utilised to randomise the plaintext using a system based on iterating the states of cellular automata.

In a further embodiment of the invention, the block cipher 20 incorporates substitution tables, which are initialised using the initialisation vector IV and external key EKEY. In a still further embodiment of the invention the above mentioned substitution tables are continuously updated to make the mapping from the input to the output varying. This is accomplished by swapping the addressed line of the decryption (incl. encryption) substitution table with another line addressed by a special field of the IKEY data. Since the mapping defined by a substitution table used in encryption must be inverted for decryption, any amendment of this mapping at set-up must be followed by the inversion of the table. In the preferred embodiment, a substitution table for decryption is generated during set-up, and then has to be inverted to obtain the substitution table for encryption. Upon decryption of the ciphertext, only the decryption substitution table need be created. This means that processing cost is reduced in decryption compared to encryption.

It is preferred that in addition to at least one substitution table, the block cipher 20 includes a plurality of parallel operations, wherein the operations executed for any particular block is determined by the internal key IKEY generated by the module 10. This may be viewed as an arrangement of parallel paths, each path being associated with a specific operation. The internal key IKEY acts as a router, sending the plaintext or partially encrypted block down one of the paths. The operations may include, but are not limited to, mapping functions, the rotation of the block and addition operations. One path may include no operation at all. This allows the function to be performed rapidly without compromising security, since

the probability that a block undergoes a specific operation depends on the number of possible paths.

It will be understood that the dimensions used for the various elements of data, for example the external key EKEY, the initialisation vector IV, and the units in which the plaintext and ciphertext are manipulated in the module 10 and the cipher system 20 are given by way of example only. It will be apparent to those skilled in the art that these values may be modified to increase the security of the cipher function or reduce the processing time for any specific operation. Furthermore, modifications may be made to the content of the module 10 for the same purpose.

As for the size of the external (secret) key EKEY note that, at least in the present exemplifying embodiment of the invention, the key could be viewed as a compressed input software module, possibly written in the module-13-language. Its recommended size should therefore preferably be in the order of several hundred bytes. This argument applies to the other inputs as well, such as the size of the input plaintext block and the size of the random Initialisation Vector IV. Note that this is implicitly included in the preferred structure of the input (Fig 4). This will be an issue mostly when using the invention in security related environments, and it should be noted that in random number applications (by example only) of the present invention, this note may be of no relevance.

Also implicitly included in the present embodiment of the invention is that the internal information paths are intentionally of different sizes in different places. Referring to Fig. 3 the flow of information IKEY from module 10 to module 20 is much higher than the flow of information through module 20. Each 32 bits of plaintext, to be encrypted by module 20, corresponds to one instance of IKEY, which, in the preferred embodiment, has the size 8 times 32 bits (or 13 times 32 bits from module 14 to module 15 in Fig. 1). This implies that each 32 bits of output from module 20 could, depending on IKEY, correspond to any 32 bit input plaintext, and that for each 32 bit output, from module 20, or possible each combination of 32 bit input and 32 bit output from module 20, could correspond to a large subset of all possible internal keys IKEY.

The man skilled in the field of the present invention will clearly see the benefit of this construction, but note that in other application areas of the present invention, such as random number generation, this may not be necessary to achieve.

An important characteristic of the arrangement and method according to the present invention is that providing an increased number of operations in the instruction table 13 will substantially improve the security of a system, or the quality of generated random numbers, by increasing the possible operations that may be performed. However, although such a modification will involve a higher initial hardware or software outlay, the delay in encryption or decryption will not be increased, if it is assumed that all instructions can be executed in approximately the same length of time. Accordingly, the security of a system may be increasing without an associated time penalty.

The inventive method may be realised by means of hardware as well as software programs executed on a computer comprising a processor, storage means and input/output means. A selected realisation comprises functional means arranged to perform the different steps of the function of the invention as has been described in the present description. An embodiment of the invention in the shape of a computer program can further be realised as a computer program product possibly comprising a storage medium and means, stored on the storage medium, for controlling a computer to perform the functions and the steps of the invention. Another embodiment of the invention is realised as a data stream or a carrier signal modulated by signals representing a computer program arranged to control a data processing apparatus, for example a computer, to perform the functions and the steps of the invention.

The present invention has been shown and described in terms of specific embodiments, but it is obvious for the person skilled in the art that different combinations and modification of features can be made without departing from the scope of the invention as described in the appending claims.